SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Bruno Grgić

Zagreb, 2015.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor:
prof. dr. sc. Bojan Jerbić

Student:
Bruno Grgić

Zagreb, 2015.

Izjavljujem da sam završni rad izradio samostalno, služeći se literaturom i znanjem stečenim tijekom studija.

Zahvaljujem se mentoru prof. dr. sc. Bojanu Jerbiću, voditelju rada na iskazanom povjerenju i korisnim savjetima.

Također se želim zahvaliti asistentu mag. ing. Filipu Šuligoju na korisnim sugestijama i pomoći pri izradi završnog rada.

Bruno Grgić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum: 27 -02- 2015	Prilog
Klasa: 602-04/15-6/3	
Ur.broj: 15-1703-15-151	

ZAVRŠNI ZADATAK

Student: **Bruno Grgić** Mat. br.: 0035185307

Naslov rada na hrvatskom jeziku: **Praćenje kontura pomoću robota**

Naslov rada na engleskom jeziku: **Robotic contour following**

Opis zadatka:

U radu je potrebno razviti upravljački i vizijski algoritam robota za praćenje vanjskih kontura nepoznatih (plošnih) predmeta. Pri razvoju vizijskih algoritama za prepoznavanje kontura koristiti OpenCV knjižnicu. Razvijene algoritme provjeriti na robotu vodeći računa o sljedećem:

- omogućiti prepoznavanje kontura nepoznatih predmeta s nepoznatom orijentacijom u radnoj ravnini robota,
- osigurati transformaciju točaka koje opisuju predmet iz koordinatnog sustava kamere u koordinatni sustav robota,
- praćenja konture predmeta temeljiti na vektoru koji opisuje vrh alata robota.

Zadatak zadan:
25. studenog 2014.

Rok predaje rada:
1. rok: 26. veljače 2015.
2. rok: 17. rujna 2015.

Predviđeni datumi obrane:
1. rok: 2., 3., i 4 . ožujka 2015.
2. rok: 21., 22., i 23. rujna 2015.

Zadatak zadao:
Prof. dr. sc. Bojan Jerbić

Predsjednik Povjerenstva:
Prof. dr. sc. Zoran Kunica

SADRŽAJ

POPIS SLIKA

SAŽETAK

Cilj ovog rada bio je razviti vizijski sustav koji prepoznaje konture željenog predmeta.

Uvod daje kratki opis potrebe za automatizacijom industrijskog brušenja i prednosti koje vizijski sustav pridonosi takvim procesima.

U prvom poglavlju razvijen je vizijski sustav baziran na Canny algoritmu i čvorovima aproksimacijskih funkcija. Prije dobivanja točaka kontura, primjenjeni su medijanski filter i piramidno skaliranje za dodatno isticanje obrisa objekata na slikama. Nakon dobivanja kontura predmeta, koristeći njihovo hijerarhijsko stablo uspjeva se pristupiti željenim konturama, te se postavljaju razni uvjeti provjere kako bi se osigurala visoka točnost prepoznavanja željenih obrisa. Vizijski sustav može se primijeniti na predmetima različitih oblika, dimenzija i orijentacije. Vizijski i upravljački algoritam razvijani su u programskom jeziku C++. Korištena je knjižnica OpenCV u Microsoft Visual Studio 2010 programskom okruženju.

Drugo poglavlje opisuje transformaciju koordinatnog sustava kamere i koordinatnog sustava robota u jednu točku radne površine. Ishodišta vizijskog sustava i koodinatnog sustava robota moraju biti precizno usklađena ako želimo točno praćenje kontura predmeta prihvatnicom robota. Obrađen je upravljački algoritam koji komunicira podatke o tim konturama robotu u obliku točaka. Točke pretvorene iz piksela u milimetre preuzima robot Ethernet komunikacijom. Programski jezik robota je Karel.

Proces simulira postupak industrijskog brušenja predmeta na proizvodnoj liniji.

1. Uvod

Vizijski sustavi integralni su dio velikog broja elektroničkih uređaja u današnjem svijetu. Padom cijene elektroničkih komponenata video kamere postale su lako dostupne, te istraživanje njihove primjene otkriva za sada neograničene mogućnosti unaprijeđenja postojećih procesa, kako u svakidašnjem životu, tako i u industrijskoj primjeni. Osobito zanimljivo područje primjene vizijskih sustava u industriji su industrijski roboti.

Umjesto unaprijed definiranih gibanja robota, dobro implementiran vizijski sustav može višestruko proširiti područje djelovanja, ali i unaprijediti trenutne radnje. U opsegu ovoga rada, istražit će se mogućnost primjene video kamere u procesu robotiziranog brušenja.

Brušenje koje izvodi čovjek je težak, fizički naporan posao. Radna okolina nije prikladna za čovjeka zbog stalne izloženosti odstranjenim česticama koje mogu narušiti zdravlje radnika. Fizičkih radnika za posao brušenja je sve manje zbog vrlo slabog interesa mlađih skupina populacije za poslove ručnog odstranjivanja čestica. Automatiziranje tog procesa ne samo da uklanja rizično zanimanje iz sfere ljudskih zanimanja, već i pruža sigurnost obavljanja poslova koji su sve više deficitirani.

Automatizirani postupak brušenja bez vizijskog sustava može naići na niz problema koje je potrebno riješiti za zadovoljavajuću razinu kvalitete uratka. Razvojem materijala, novih postupaka oblikovanja i izrade predmeta, predmeti koje je potrebno brusiti su sve složenijih oblika. Materijal može biti nedovoljno istražen, a njegova tekstura neujednačena kao što je slučaj kod nekih legura i kompozita. Vizijski sustav bi u tim slučajevima uvelike povećao ponovljivost procesa, a samim time i osigurao konstantnu kvalitetu.

U prvom dijelu rada opisan je razvoj vizijskog sustava za praćenje vanjskih kontura nepoznatih plošnih predmeta. Sustav je testiran na raznim oblicima površine, koji bi mogli zamijeniti oblike lima za brušenje. Prethodno pronalaženju kontura primjenjene su razne funkcije filtriranja, manipulacije kontrasta i boja slike kako bi se olakšalo izdvajanje obrisa. Canny algoritam se pokazao kao pouzdana metoda pronalaženja obrisa predmeta. Oni su izdvojeni u strukturu kojom je moguće manipulirati hijerarhijama i aproksimacijskim funkcijama koje nam olakšavaju specificiranje pojedinačnih kontura. Postavljanjem dodatnih uvjeta provjere, rad vizijskog sustava može se dodatno osigurati od pogrešaka.

Funkcije filtriranja i obrade slike, rad Canny algoritma, funkcije aprosksimacije krivulja i čvorova objašnjene su i popraćene softverskom implementacijom. Navedeni su rezultati vizijskog algoritma u raznim uvjetima rada, koja su njegova ograničenja i kako bi se mogao dodatno unaprijediti.

Nakon toga testiran je i transformiran koordinatni sustav kamere, usklađen sa koordinatnim sustavom robota i primijenjen upravljački algoritam za komunikaciju robota i vizijskog sustava.

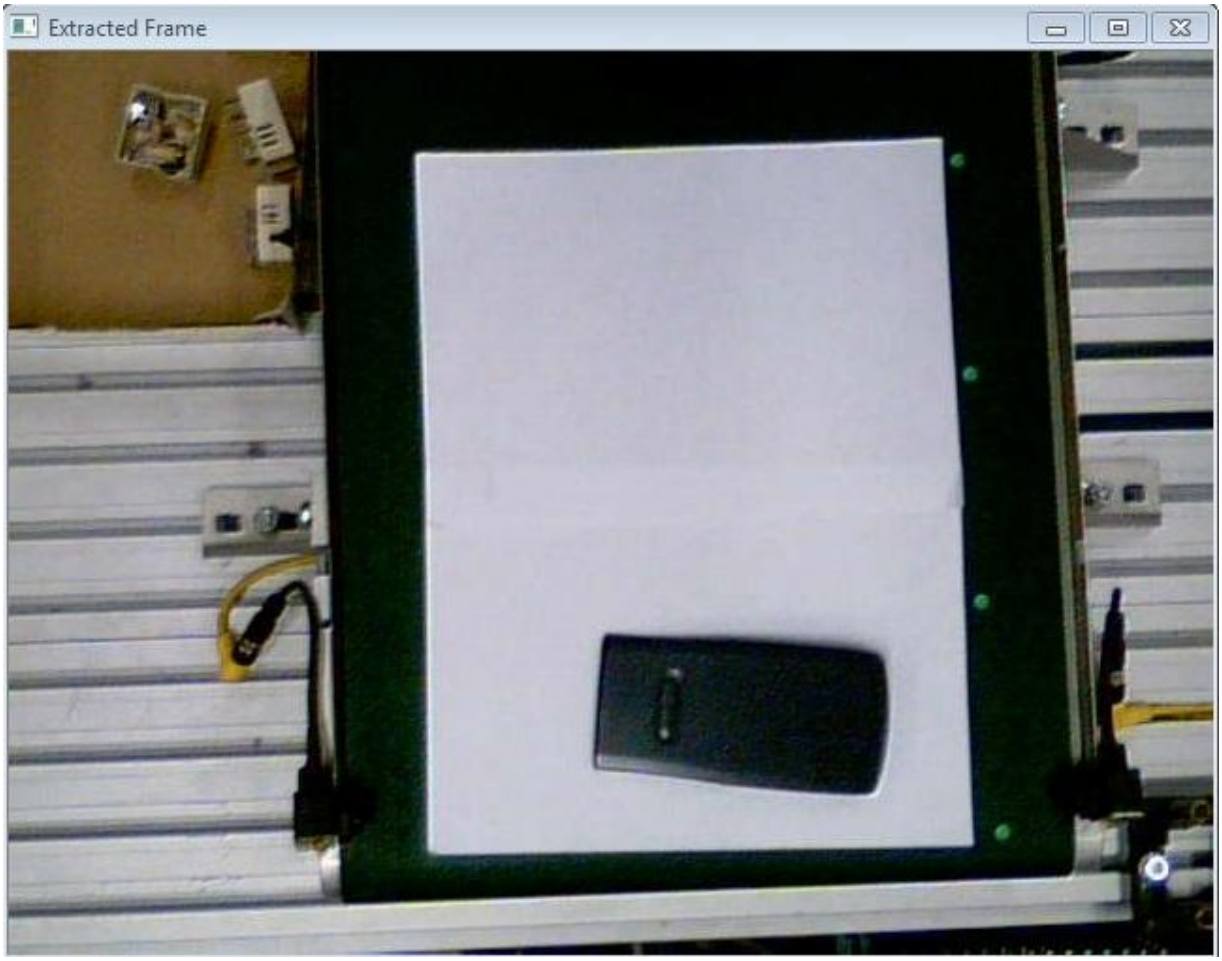
Komunikacija se vršila preko TCP / IP protokla, dok su točke kontura robotu proslijeđene programom napisanim u programskom jeziku Karel. Robot za koji je vizijsku sustav predviđen je Fanuc M3-iA. Na njega je montirana prihvatnica sa standardnom hvataljkom koja bi u pravoj primjeni bila zamjenjena radnim alatom za brušenje.

Slika 1. Fanuc M3-iA

Slika 2. Primjer radnog alata

2. Razvoj vizijskog algoritma

Za vizijski sustav nam je prije svega potrebna video kamera kako bi mogli analizirati sliku i izvršiti obradu slike. Kako bi se razvio algoritam za vizijski sustav, obradu slike i pronalaženje kontura željenih predmeta potrebno je odabrati radnu površinu koja ima specifična obilježja. U ovom slučaju to je bila površina pravokutnog oblika bijele boje. Odabrani su papiri A4 i A3 dimenzija, ali može biti i bijela površina drugih dimenzija.



Slika 3. Radna površina

Korištena je kamera Microsoft Lifecam XV-1000 montirana vertikalno iznad radne površine.

Slika 4. Kamera Lifecam XV-1000

OpenCV je biblioteka otvorenog koda koja služi razvoju aplikacija iz područja računalnog vida. Biblioteka je dostupna na Windows, Linux i MacOS X operacijskim sustavima, te ju je moguće koristiti u programskim jezicima C i C++, dok su za Python, Matlab, Octave, Ruby, C# i druge, dostupna programska sučelja (eng. wrapperi).

Uz pomoć OpenCV biblioteke moguće je razvijati aplikacije za obradu slike u stvarnom vremenu, čitanje, stvaranje i obradu video sadržaja iz datoteka ili kamera, strojno učenje, te projekcije i 3D prikaze.

Osnovna zaglavlja biblioteke su *cv.h*, *highgui.h* i *ml.h*. Zaglavlje *cv.h* sadrži, opis osnovnih struktura podataka, definicije konstanti i zastavica, te osnovne funkcije za obradu slike. U zaglavlje *highgui.h* opisane su funkcije i strukture podataka za prikaz sadržaja, te izradu korisničkog sučelja.

Program iz ovog završnog rada razvijen je u programskom jeziku C++, na operacijskom sustavu Windows 7, uz korištenje OpenCV 2.4.10 biblioteke.

1. Instalacija OpenCV knjižnice

Da bi se mogla koristiti OpenCV knjižnica potrebno ju je integrirati u programsko okruženje. U ovom slučaju to je bio Microsft Visual Studio 2010. Slijede kratke upute za postavljanje:

1. Instalacija u C:

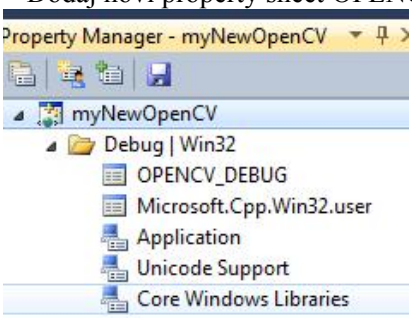
C:\opencv248\build

2.ENVIRONMENT VARIJABLE

OPENCV_BUILD=C:\opencv248\build
Path=;%OPENCV_BUILD%\x86\vc10\bin;

3.RESTART Windows-a

4.VISUAL STUDIO 2010 Ultimate– property manager (View – show additional windows)
- Dodaj novi property sheet OPENCV_DEBUG

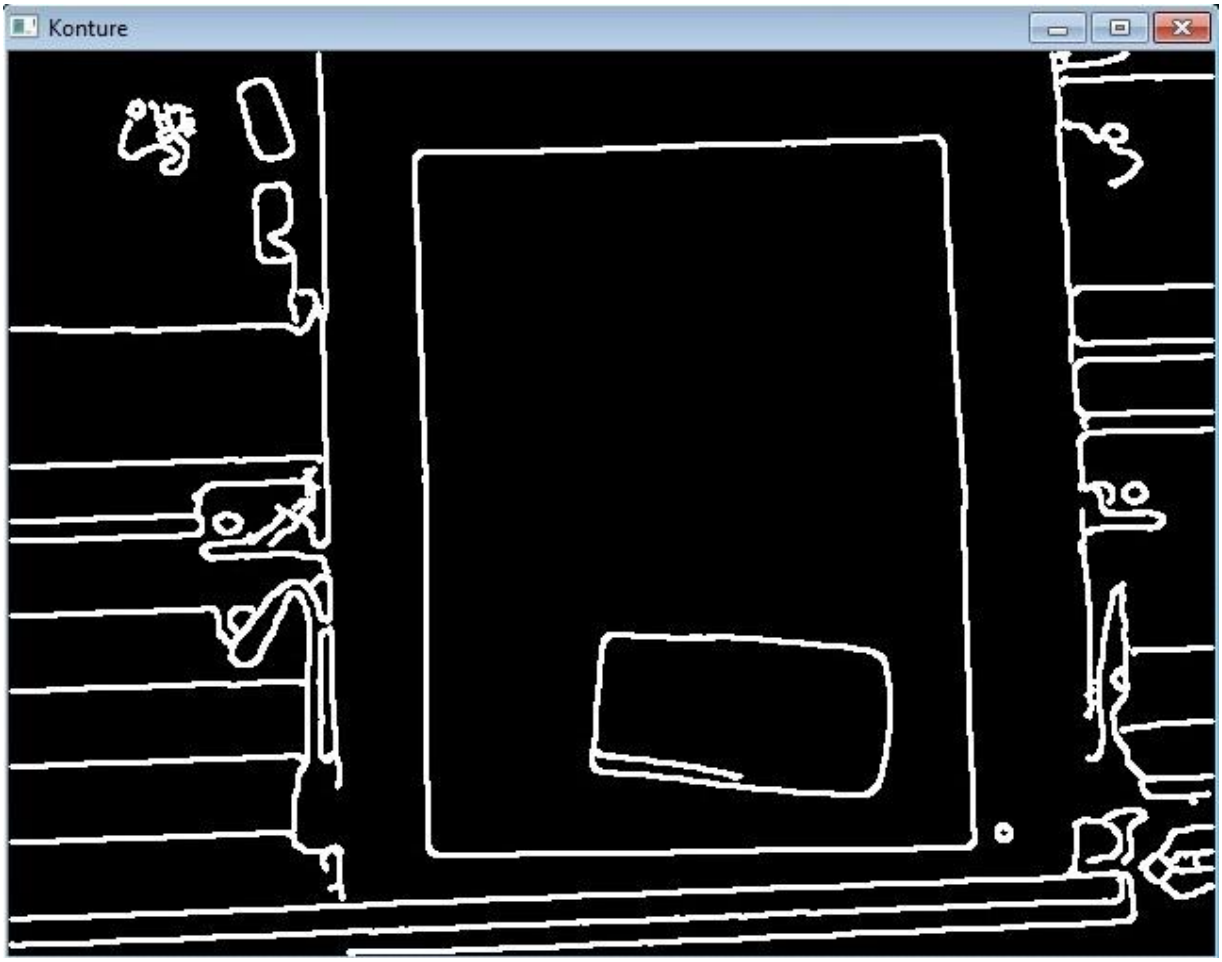


a) C/C++- General – aDtitional include directories - **$(OPENCV_BUILD)\include**
b) Linker – General – Aditional Liibrary directories **- $(OPENCV_BUILD)\x86\vc10\lib**
c) Linker – Input – Additional dependencies (svi od tuda samo sa d na kraju C:\opencv248\build\x86\vc10\lib)
    a. opencv_calib3d248d.lib
    b. opencv_contrib248d.lib
    c. opencv_core248d.lib
    d. opencv_features2d248d.lib
    e. opencv_flann248d.lib
    f. opencv_gpu248d.lib
    g. opencv_highgui248d.lib
    h. opencv_imgproc248d.lib

2. Pokretanje algoritma

Video kamera snima sliku u određenom broju “sličica” po sekundi, tako da se obrada slike odvija na svakoj slici zasebno. Razvijen je algoritam koji omogućava zaustavljanje snimanja slike, kako bi se mogla izvršiti obrada i vidjeti njeni rezultati, te nakon nje i upravljački algoritam koji komunicira sa samim robotom za izvršavanje gibanja.

Osim ograničenja rezolucije kamere i njene leće, šumova slike, promjenjivih uvjeta osvjetljenja, uz radnu površinu na snimljenoj slici nalazi se i veći broj drugih objekata koji otežavaju pronalaženje konture ciljanog predmeta. Te poremećaje je potrebno eliminirati kako bi vizijski sustav obavljao željenu radnju.



Slika 5. Konture objekata snimljene slike

Slijedi objašnjenje algoritma vizijskog sustava. Koraci će biti popračeni objašnjenjem koda, njegove primjene i koje su bile prepreke tokom implementacije.

U zaglavlju programa je prije svega potrebno aktivirati dijelove OpenCV knjižnice koji se koriste unutar programa.

```
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv/cv.h>
```

Prva funkcija koja se poziva u svakom programu je funkcija *main*. Njeni argumenti su unosi korisnika, čiji je tip definiran kao broj, tj. tip podatka integer. Unos će zapravo biti slova za manipuliranje radom algoritma, ali treba uzeti u obzir da slova prevedena preko ASCII koda su brojevi.

Za početak rada bilo kojeg algoritma obrade slike neophodno je imati unos podataka u obliku slike.

Klasa VideoCapture definira se kao klasa "capture" za lakše pozivanje funkcija unutar klase. Pozivanjem člana funkcije *open,* operatorom "." otvara se video datoteka za čitanje slike ili aktivira kamera koja šalje video prikaz. Broj "0" definira vrstu kamere spojene na računalu, u ovom slučaju to je vanjska. Nakon toga radi se provjera koja gasi program ako video ne postoji.

```
int main(int argc, const char** argv)
{

    cv::VideoCapture capture;
    capture.open(0);
    if (!capture.isOpened())
        return -1;
…
```

Algoritam radi unutar beskonačne petlje koja se zaustavlja ili prekida pod određenim uvjetima.

Uvjete zaustavljanja ili prekidanja petlje regulira korisnik programa, koje u pozadini odrađuje funkcija switch. Ona radi kao sklopka, provjeravajući koji je uvjet ispunjen, te u slučaju zadovoljavanja određenog uvjeta vrši radnju. Ovdje se provjera uvjeta temelji na unosu korisnika koji je pohranjen u varijablu *c*.

```
c = cvWaitKey(10);
c = tolower(c);
    if(c == 27 || c == 'q')
       break;

    switch( c )
    {
    case 'p':
       stop = !stop;
       break;
    case 's':
       singlestep = true;
       stop = false;
                        once1 = true;
                        once2 = true;

       break;
    case 'r':
       stop = false;
       singlestep = false;
       break;
              }
```

Preuzimanje slike zaustavlja se promjenom Bool varijable *stop*. Ona je u odjeljku sa varijablama definirana kao neistinita vrijednost *false*, tako da *if* naredba nastavlja sa izvršavanjem koda samo ako je varijabla *stop* neistinita.

Dakle, tipka *p* na tipkovnici ima funkciju pauze mijenjajući Bool vrijednost svakim njenim pritiskom.

Pritiskom *r* resetiraju se varijable na početne vrijednosti, a tipkom *s* upravlja se radom obrade slike.

Obrada se vrši isključivo ako je Bool varijabla *singlestep* istinita što će se vidjeti u nastavku.

Unosom *q* ili *esc*, poziva se funkcija *break*, koja prekida beskonačnu *for* petlju izvan koje se nalaze naredbe za gašenje svih prozora i preuzimanja slike.

```
for(;;)
{
        if ( !stop )
        {
                rawImage = capture.read(Frame);
                if (!rawImage)
                        break;
        }
        cv::imshow( "Extracted Frame",Frame);
        …
}
capture.release();
cv::destroyAllWindows;
```

Glavni uvjet nastavka rada petlje je da postoje podaci u matrici *Frame*.

Član funkcija klase *capture.read* sprema sliku u navedenu matricu i istovremeno vraća Bool nazvan *rawImage*. U slučaju da je matrica prazna, Bool će biti neistinit, te će se naredba *if* koja prekida program izvršiti.

Funkcija *imshow* prikazuje matricu *Frame* u prozoru *Extracted Frame*. Ona je u ovom slučaju direktna snimka sa videokamere jer na njoj nije odrađena nikakva obrada. Direktna snimka

videokamere vrlo je korisna tokom testiranja vizijskog sustava. Preporučljivo je imati ju paralelno aktiviranom ne samo tokom testiranja sustava, već i za osmišljavanje rješenja, jer svaka okolina predstavlja različite prepreke u izvršavanju obrade slike kakvu želimo.

Matricu *Frame* i prozor *Extracted Frame* nužno je definirati kao varijable prije pokretanja samog programa.

Prije nego što je započeta obrada slike zaustavlja se preuzimanje video zapisa sa kamere kako se ne bi nepotrebno punila memorija računala.

```
…
if ( singlestep )
        stop = true;
```

Slijedi provjera postoji li ulaz (video zapis sa kamere) i uvjet da je slika zaustavljena varijablom *singlestep*.

```
if( rawImage && singlestep)
{
Početak obrade slike
}
```

3. Obrada slike prije traženja kontura

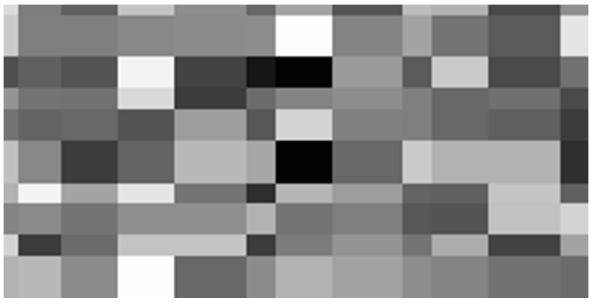
Funkcijama *pyrDown* i *pyrUp* cilj je dobiti uglađenu sliku što manjim brojem smetnji za algoritam traženja kontura. *pyrDown* smanjuje sliku, izjednačava susjedne piksele čime je rezolucija te slike smanjena. Za traženje kontura ujednačenost piksela rubova predmeta i eliminacija okolnih smetnji bitniji su od rezolucije slike. *pyrUp* funkcija vraća veličinu slike na izvornu, popunjavajući nedostajuće pixele sa nulama. Time smo dobili oštrije prijelaze između susjednih objekata.

```
cv::pyrDown(Frame, pyr, cv::Size(Frame.cols/2, Frame.rows/2));
cv::pyrUp(pyr, Image , Frame.size());
```

Kako bi lakše uočavali konture vrlo je korisno RGB sliku pretvoriti u crno bijelu. Tim postpukom povećavamo kontrast i omogućavamo lakše upravljanje pragom Canny algoritma koji će u nastavku biti detaljnije opisan.

```
cv::cvtColor(Image, Image_Gray, CV_BGR2GRAY);
```

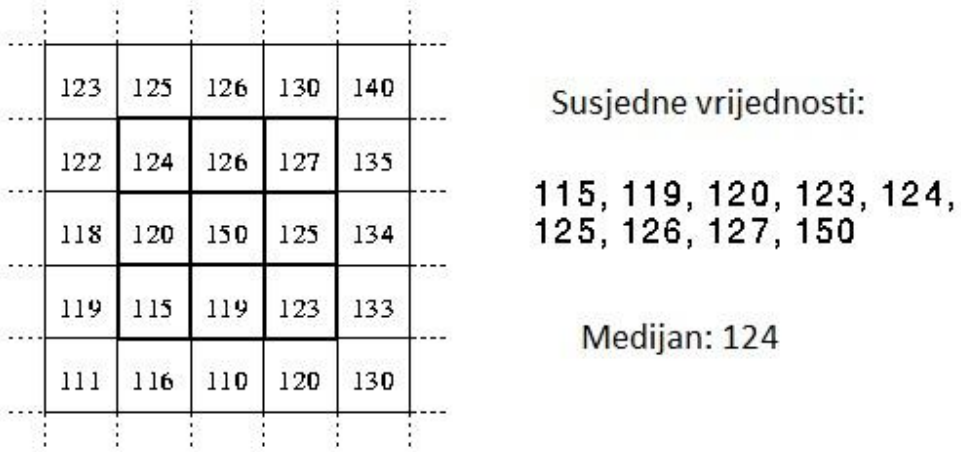
Na crno bijelu sliku primjenjen je *median filter*. Razlog tome je što stvarni signali imaju odstupanja od idealnog signala kakvog bi htjeli proizvesti našim modelom za stvaranje signala. Ta odstupanja (šum) nisu dio idealnog signala i mogu nastati iz više izvora. Primjeri su: varijabilnost osjetljivosti senzora, vremenski uvjeti, diskretna narav zračenja, greška prijenosa i diskretizacije signala.



Slika 6. Šumovi signala

Median filter razmatra redom svaki piksel u slici i uspoređuje ga sa susjednim pikselima kako bi utvrdio pripada li okolini. U slučaju da ne pripada, ne pridjeljuje mu srednju vrijednost susjednih piksela, već mu dodjeljuje medijan susjednih vrijednosti. Medijan se računa tako da se numerički sortiraju vrijednosti svih piksela iz neposredne blizine i odabere iznos koji je u sredini niza. Medijan je manje osjetljiv na ekstremne vrijednosti od aritmetičke sredine, što ga čini posebno pogodnim za nepravilne asimetrične distribucije. U slučaju da postoji parni broj susjednih piksela, računa se srednja vrijednost 2 piksela u sredini niza.

Time su na optimalan način uklonjeni šumovi sa slike uz zadržavanje čim većeg broja korisnih detalja slike.



Slika 7. Medijan različitih točaka

4. Detekcija rubova

1. Canny detektor

Jednu od metoda detekcije rubova razvio je John F. Canny 1986. Ona koristi algoritam s 5 faza da bi detektirala rubove u raznim slikama.

Canny je razvio teoriju koristeći analitički pristup na osnovi 1-D kontinuiranog modela skokovitog ruba iznosa h_E te aditivnog Gaussovog šuma varijance s_n .

Detekcija se obavlja konvolucijom 1-D impulsnog odziva $h(x)$ i zašumljenog signala $f(x)$, a rub se označava na mjestu maksimalnog odziva u konvoluciji signala $h(x)$ i $f(x)$.

Maska $h(x)$ je odabrana tako da zadovoljava tri kriterija:

1. Dobru detekciju (maksimizacija odnosa signal/šum kod odziva)
2. Dobru lokalizaciju (izbjegavanje pogrešnog označavanja mjesta ruba)
3. Jedan odziv (potrebno je da postoji samo jedan detektirani rub)

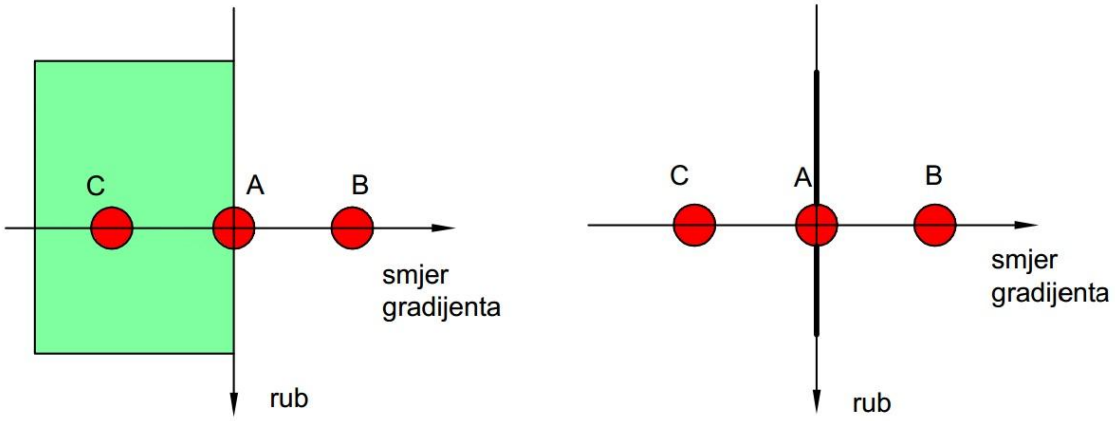
Faze algoritma:

1. Kreirati 1D Gausovu masku $h(x)$
2. Kreirati 1D maske prve derivacije Gausove funkcije po x i y $h(x)_x$ i $h(x)_y$
3. Konvoluirati sliku s $h(x)$ po x i y smjeru (dobije se I_x i I_y)
4. Konvoluirati I_x sa $h(x)_x$ i I_y sa $h(x)_y$ (dobije se I'_x i I'_y)
5. Izračunati amplitudu gradijenta

$$A(x,y) = \sqrt{I'_x(x,y)^2 + I'_y(x,y)^2}$$

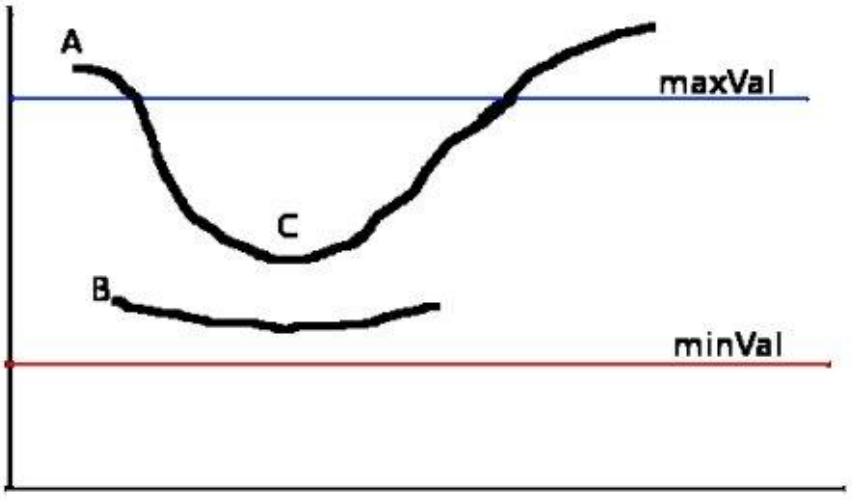
Uspoređivanje vrijednosti modula gradijenta sa zadanim pragom i zadržavanje samo onih piksela kod kojih je modul gradijenta veći od zadanog ne daje dobre rezultate, te se radi potiskivanje ne-maksimuma (eng. non-maximum supression). Za svaki piksel slike izračunati

modul gradijenta se usporedi sa vrijednostima modula gradijenta u dvije točke koje leže simetrično sa obje strane ruba na pravcu gradijenta za promatrani piksel. Samo ako je vrijednost modula gradijenta u danom pikselu veća od obje vrijednosti sa kojima se uspoređuje, taj gradijent se zadržava, dok se u protivnom izjednačava sa nulom. Na ovaj način se aproksimativno određuje maksimum prvog izvoda u pravcu normalnom na pravac ruba.



Slika 8. Canny detektor

Nakon potiskivanja ne-maksimuma primjenjuje se postupak uspoređivanja sa pragom radi formiranja binarne mape rubova, gdje se utvrđuje koji rubovi su uistinu rubovi objekata. Potrebne su nam 2 vrijednosti praga, označene na slici 8 kao *maxVal* i *minVal*. Bilo koji rub sa vrijednosti gradijenta iznad *maxVal* smatra se rubom, dok svi rubovi vrijednosti gradijenta ispod *minVal* su odbačeni.



Slika 9. Histereza detekcije kontura

Rubovi koji su između te 2 vrijednosti imaju mogućnost biti rub i biti odbačeni, a uvjet je da spoj sa već klasificiranim rubom. Vrijednosti *maxVal* i *minVal* u kodu zastupljenje su varijablama *tresh*, *tresh*4.* Njihove vrijednosti mogu biti od 0 do 255, gdje 0 predstavlja crnu boju, a 255 bijelu boju.

```
Canny( Image_Blur, canny_output, thresh, thresh*4, 3 );
```

Ulazna matrica je *Image_Blur,* tj. izlazna slika median filtera, dok je izlaz jednokanalna 8-bitna slika detektiranih rubova *canny_output*. Pikseli različiti od nule tretiraju kao vrijednost 1, a pikseli jednaki nuli kao 0. Slijedi pronalaženje kontura i njihova hijerarhija iz izlaza Canny operatora.

2. Hijerarhija kontura

Sljedeća funkcija *findContours* pronalazi konture predmeta iz binarne slike rubova. Bazirana je na algoritmu Suzuki[85] objavljenom u radu "Suzuki, S. and Abe, K., Topological Structural Analysis of Digitized Binary Images by Border Following".

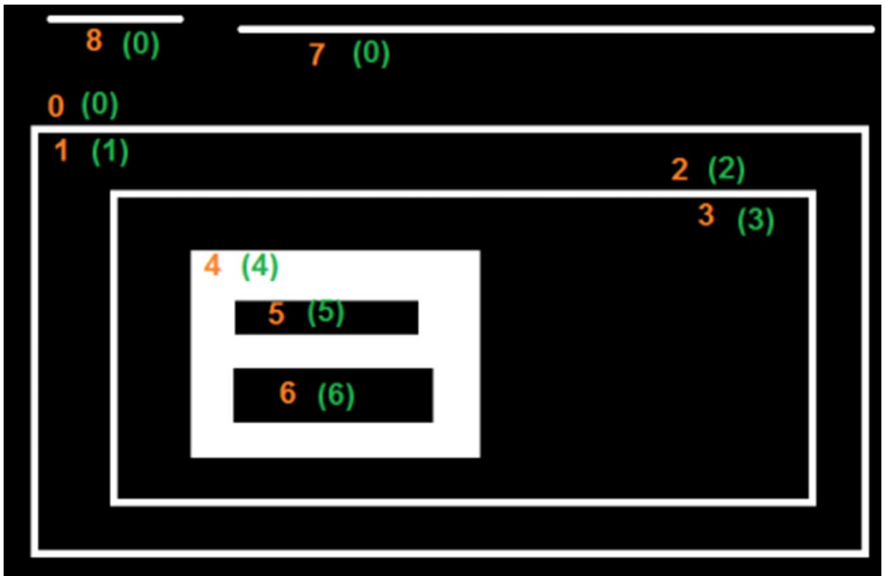
```
findContours( canny_output, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, cv::Point(0, 0) );
```

Osim samih kontura, koje su pohranjene u vektoru *contours* funkcija ima niz korisnih parametara i izlaza. Vrlo bitna je hijerarhija kontura pohranjena u vektoru *hierarchy*. Strukturirana je kao gnijezdo kontura. U tom gnijezdu jednoznačno su definirani roditelj i potomak svake konture. Roditelj je kontura unutar koje su smještene druge konture, dok je potomak kontura smještena unutar neke konture. Na taj način uspostavljen je međuodnos kontura koji nam omogućava pristup specifičnim konturama i njihovim koordinatama. U nastavku će biti opisano kako se koristila hijerarhija za rješavanje problema označavanja konture na radnoj podlozi.

Svaki član vektora hijerarhije predstavljen je sa 4 vrijednosti za svaku konturu iz vekotra *contours*:

[**hierarchy[i][{0,1,2,3}]**

1. član je sljedeća kontura u istoj razini gnijezda
2. član je prethodna kontura u istoj razini gnijezda
3. prvo dijete konture
4. roditelj konture



Slika 10. Hijerarhijsko stablo

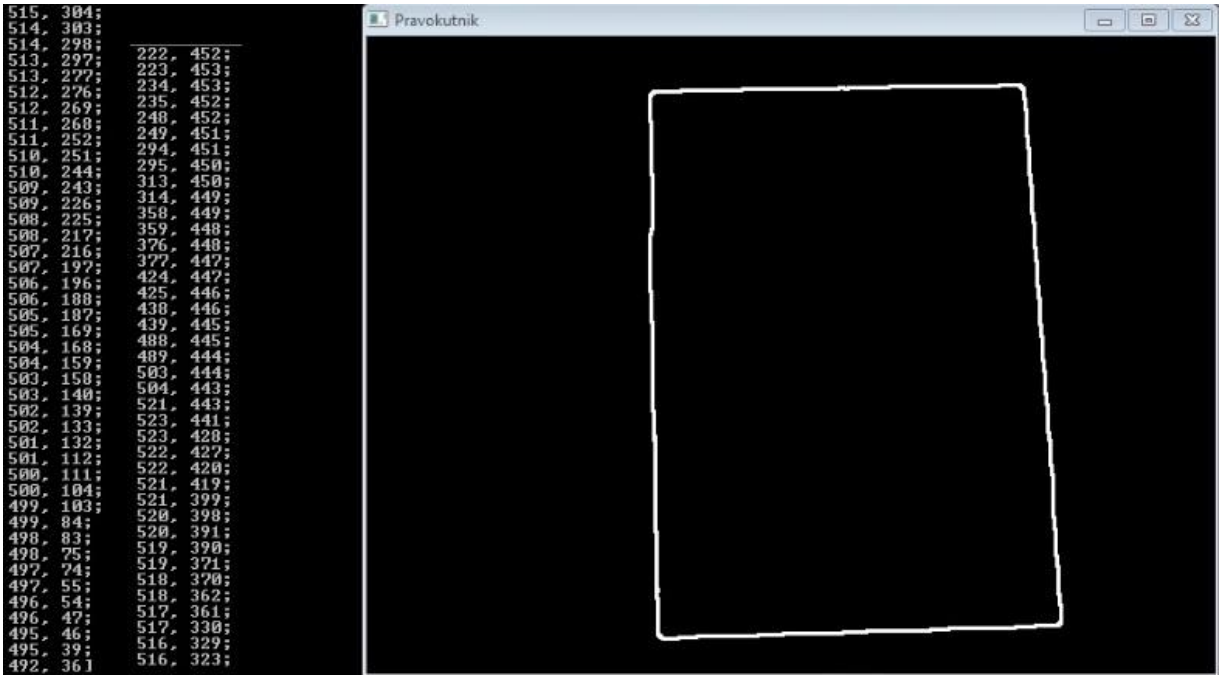
Na slici možemo vidjeti stablo hijerarhija u kojemu svaka kontura ima svoju razinu, te u kakvom je odnosu sa drugim konturama. Ako ne postoji kontura koja je u određenom odnosu sa nekom konturom, tada će taj član vektora u hijerarhijskom zapisu biti jednak "-1".

Primjer su konture 7 i 8 koje nemaju roditelje, pa je zapis za konturu 7; hierarchy[7] = {8,0,-1,-1}, te za konturu 8: hierarchy[8] = {-1, 7, -1, -1}. U toj razini hijerarhije samo kontura 0 ima dijete.

Treba uzeti u obzir da vanjski i unutarnji rub konture nisu ista kontura, već je vanjski rub roditelj unutarnjem rubu.

Da bi dobili takav zapis hijerarhijskog stabla u funkciji *findContours* nužno je definirati način dobivanja kontura kao *CV_RETR_TREE*.

Funkcijom *findContours* moguće je pohraniti sve točke kontura u vektor *contours*. Toliki broj točaka u ovom slučaju se htio izbjeći jer bi se dobio preveliki broj točaka obrisa predmeta. Komunikacija tih točaka robotu bila bi otežana, a gibanje ne bi bilo fluidno. Iz tih razloga odabrana je metoda aproksimacije *CV_CHAIN_APPROX_SIMPLE* koja kompresira horizontalne, vertikalne i dijagonalne isječke u njihove rubne točke što je vrlo praktično za gibanje robota bez narušavanja točnosti. Ipak, pokazat će se da broj čvorova ovom metodom nije uvijek dovoljno reduciran za sve potrebe, za što će nam poslužiti dodatna metoda aproksimacije.



Slika 11. Broj točaka pravokutne konture

Za postupak testiranja algoritma korištena je funkcija *drawContours*. Ona iscrtava pronađene konture iz vektora *contours* na matricu definiranu nulama *drawing*. Korištena je opcija da se iscrta specifična kontura *i*, boje *color*.

```
for( int i = 0; i< contours.size(); i++ )
    {
    drawContours( drawing, contours, i , color, 2, 8, hierarchy, 0, cv::Point() );
```

Kako se unutar programa petlja odvija po svim konturama unutar vektora, iscrat će se sve konture.

Problem je bio kako pronaći broj *i-te* konture koja predstavlja obrise predmeta na radnoj površini. Svakako je bilo moguće postaviti uvjete u petlji koja prolazi kroz vektor kontura da izdvoji traženu konturu ako već znamo neke karakteristike tražene konture. U slučaju da je pravilnog geometrijskog oblika, mogli bismo aproksimirati čvorove kontura i tražiti jednostavne geometrijske oblike, poput trokuta, pravokutnika ili kruga. Druga opcija bila je površina konture. Ali cilj ovog vizijskog sustava bio je pronaći obrise predmeta raznih geometrijskih oblika, nedefiniranih orijentacija i veličina.

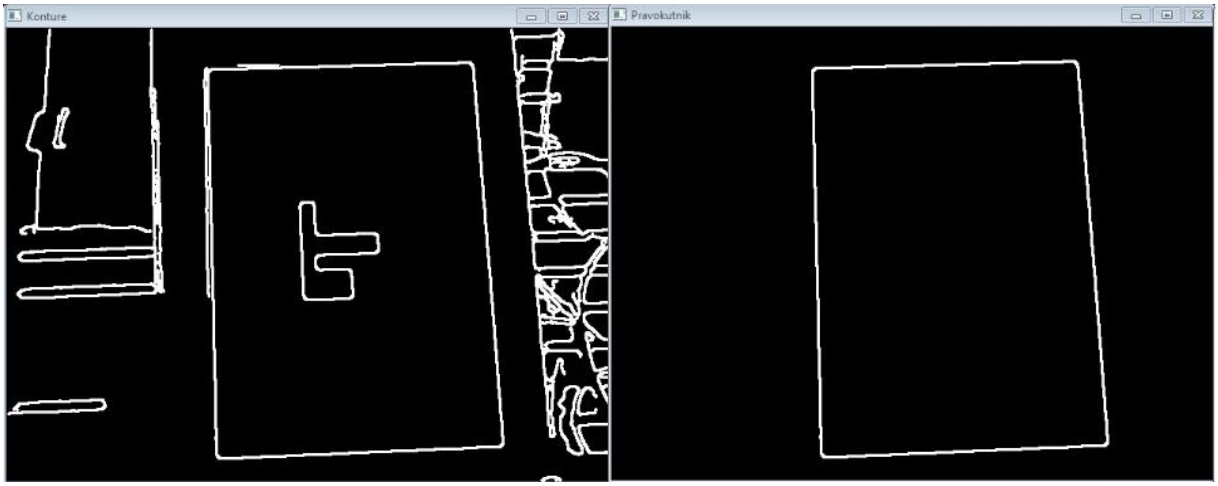
Ako se uzme u obzir da se za svaku konturu mogu očitati njeni hijerarhijski odnosi sa drugim konturama, utvrdilo se da je najpouzdaniji način prvo pronaći konturu radne površine. Tada se obrise predmeta može tražiti unutar konture radne površine. Radi šumova, ograničene kvalitete reprodukcije slike korištene web kamere i aproksimacijske greške algoritma za konture, kontura radne površine ima veći broj čvorova nego što želimo. Cilj je bio reducirati broj čvorova na 4.

Kao rješenje za pristup konturi radne površine implementiran je za Douglas-Peucker algoritam u obliku funkcije *approxPolyDP*. On reducira broj čvorova krivulje koja je aproksimirana nizom točaka. Radi to na način da u nizu točaka između prve i zadnje točke eliminira sve točke koje su na udaljnosti manjoj od definiranog iznosa. Ako se točka nalazi na većoj udaljenosti, algoritam ju spaja sa novom krivuljom. Točnost aproksimacije može se podesiti mijenjanjem varijable *epsilon*, ovdje iznosa cv::arcLength(cv::Mat(contours[i]), true)*0.02 .

```
cv::approxPolyDP(cv::Mat(contours[i]), approx, cv::arcLength(cv::Mat(contours[i]), true)*0.02, true);


if (approx.size() == 4 && (std::fabs(cv::contourArea(contours[i])) > 50000))
                {
                if (vtc == 4 && mincos >= -0.15 && maxcos <= 0.3)
                        {
```

Budući da *for* petlja prolazi kroz svaku konturu, traži se kontura čija aproksimacijska krivulja posjeduje 4 čvora u izlaznoj varijabli *approx*. Na slici koja se obrađuje nepouzdano je postaviti samo uvjet 4 čvora, jer mogu postojati konture iz okoline koje zadovoljavaju taj uvjet. Dodana je dodatna provjera površine konture, koja mora biti veća od 50000 pikela, te provjera kutova konture. Ne traži se od konture da sadrži kutove 90° radi nesavršenosti aproksimacije i šumova, pa je dovoljno da kutovi imaju približne vrijednosti pravoga kuta koji su dostatni da eliminiraju konture sa 4 čvora nepravilnih oblika.



Slika 12. Izdvajanje radne površine

Rezultat je relativno čisti obris radne površine. Kada su se zadovoljili uvjeti pronalaska konture radne površine, u petlji konture *i*, pristupilo se hijerarhiji *i-te* konture i došlo do obrisa predmeta.

3. Implementacija

Definirana je varijabla *chedo* kao dijete konture *i*.

```
int chedo = hierarchy[i][2];
```

Ona je zapravo cjeloviti broj, koji nam govori koja je to kontura unutar *contours* vektora.

Da bi se došlo do konture predmeta na radnoj podlozi, potrebno je dublje ući u hijerarhijsko gnijezdo.

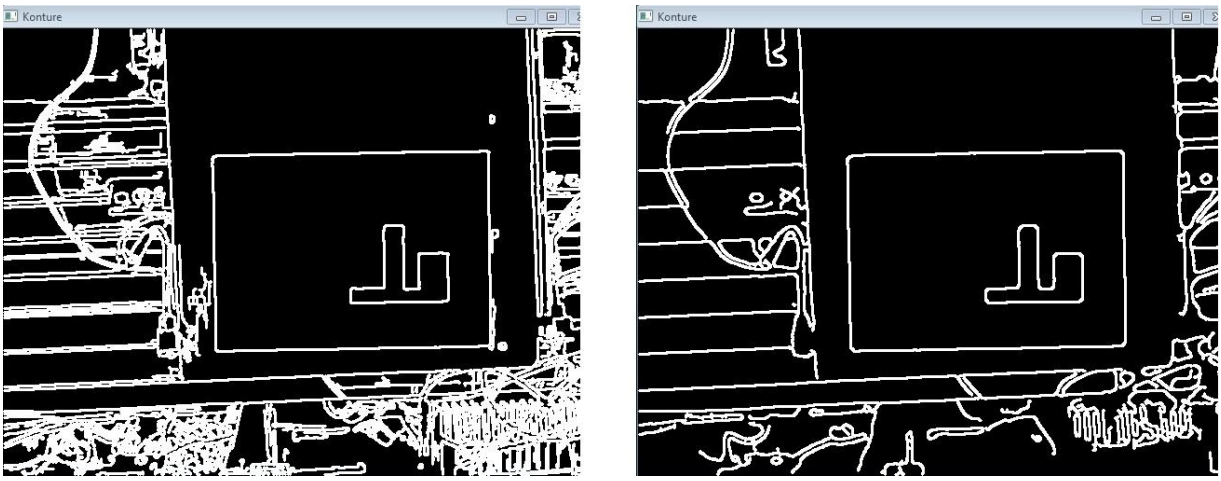
```
if ( hierarchy[chedo][2] < 0)
                    {
                        drawContours( predmet, contours, chedo , color, 2, 8, hierarchy, 0, cv::Point() );
                    }
                    else
                    {
                        drawContours( predmet, contours, hierarchy[chedo][2] , color, 2, 8, hierarchy, 0, cv::Point() );
```

Kako se traži dijete kontura radne površine, bitan uvjet koji tražena kontura mora zadovoljavati je da ona sama nema dijete konturu, tj. da niti jedna druga kontura nije ugnježđena unutar nje. Brojanje elemenata u članu vektora hijerarhije počinje od 0, pa je dijete kontura zastupljena vrijednošću [hierarchy][chedo][2]. Taj uvjet je zadovoljen ako je vrijednost 3. elementa u vektoru -1, što je svakako < 0.

U slučaju da nam je aproksimacijska funkcija *approxPolyDP* proslijedila vanjsku konturu radne površine, postavljen je *else* uvjet koji ide za još jednu razinu dublje u hijerarhijsko stablo i radi provjeru da ta kontura nema potomka. Sada kada smo pronašli traženu konturu, navest će se nekoliko primjera rada vizijskog sustava za prepoznavanje željenih kontura.

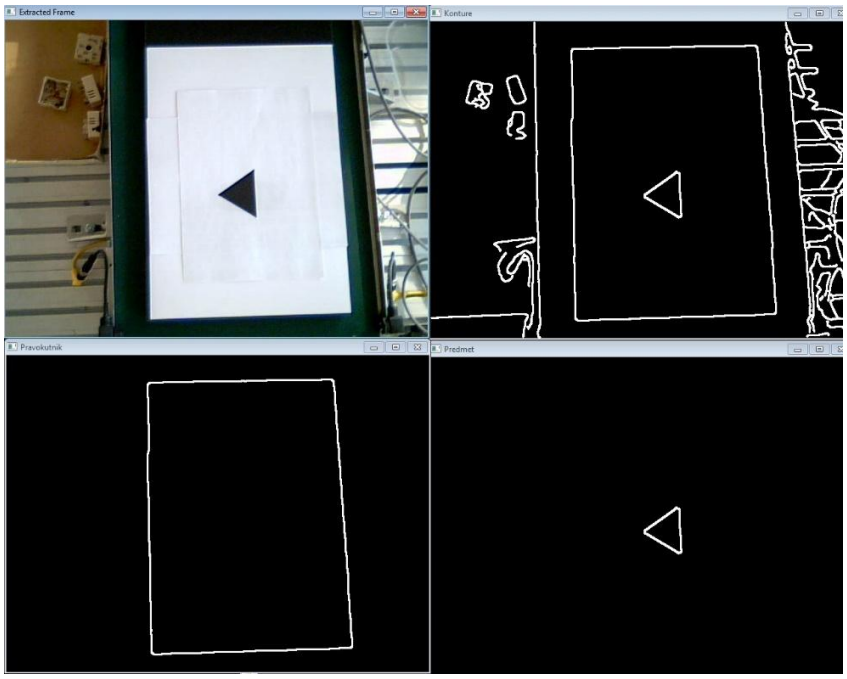
4. Rezultati

U ovom odjeljku prikazani su rezultati implementacije vizijskog sustava. Od primjera uspješnog pronalaženja željenih kontura, navedena su ograničenja rada sustava, i primjer kako se sustav nadogradio za detekciju više objekata na radnoj površini.

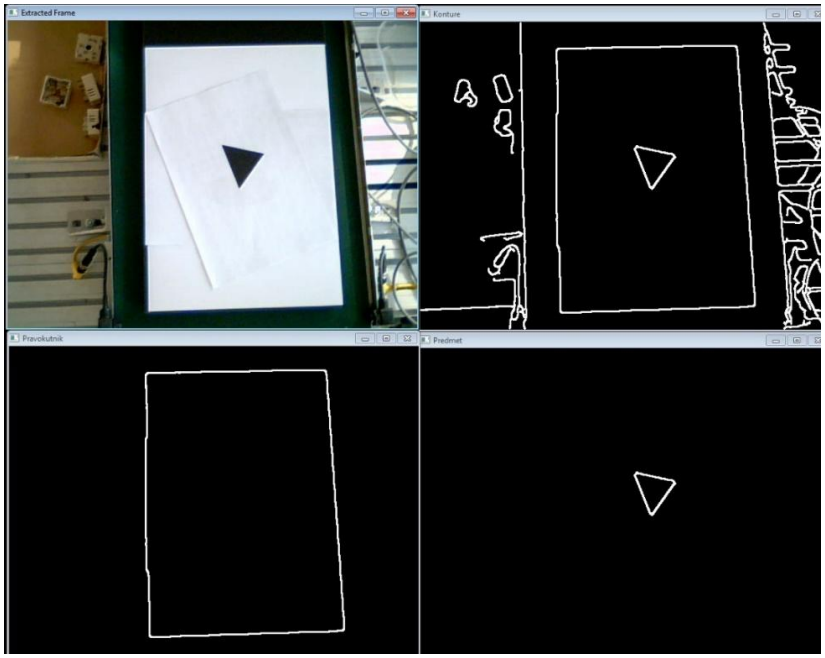


Slika 13. Usporedba kontura bez filtera i sa filterima

Slijede primjeri pronađenih kontura počevši od jednostavnijih oblika poput trokuta:

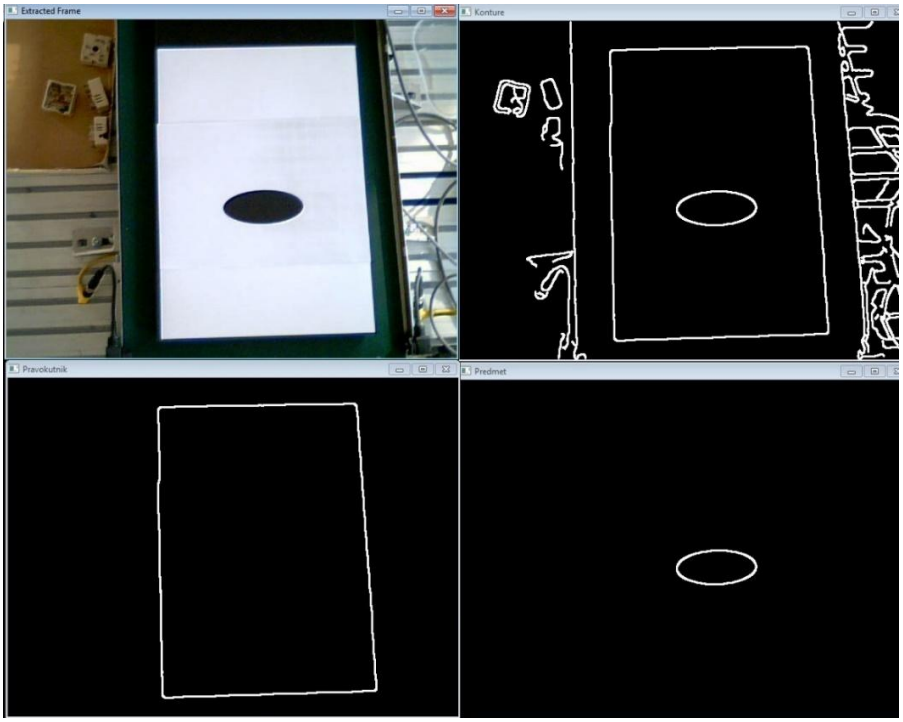


Slika 14. Trokut, orijentacija 1



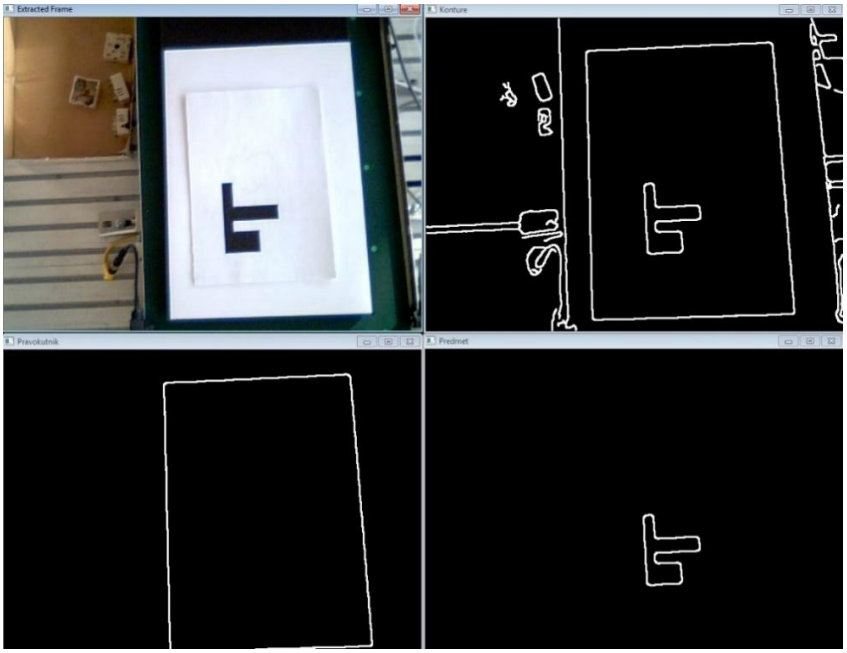
Slika 15. Trokut, orijentacija 2

Do elipsa i nedefiniranih profila:

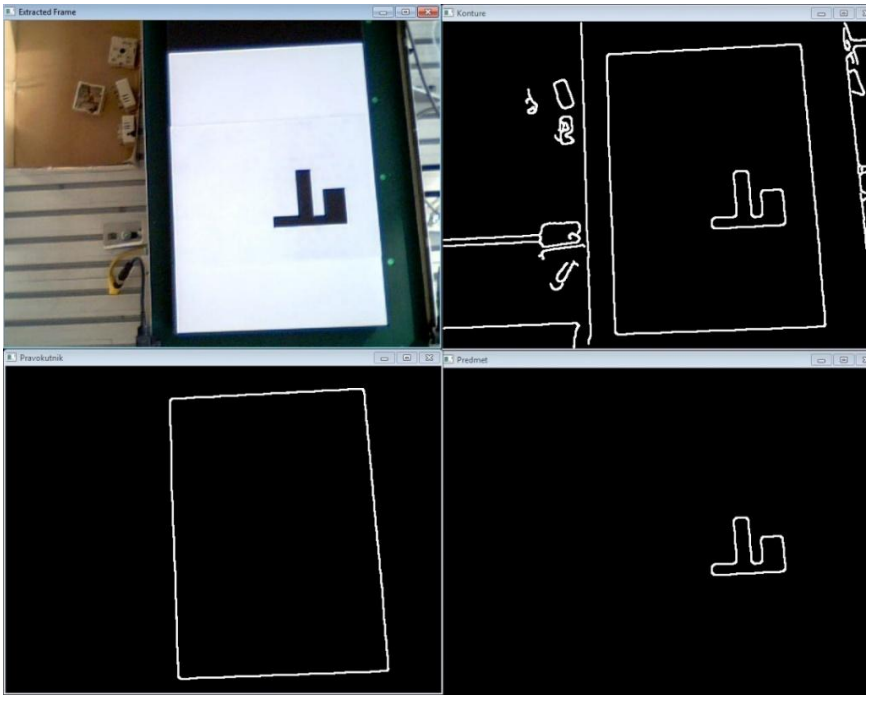


Slika 16. Kontura elipse

Nedefinirani profil koji predstavlja lim:

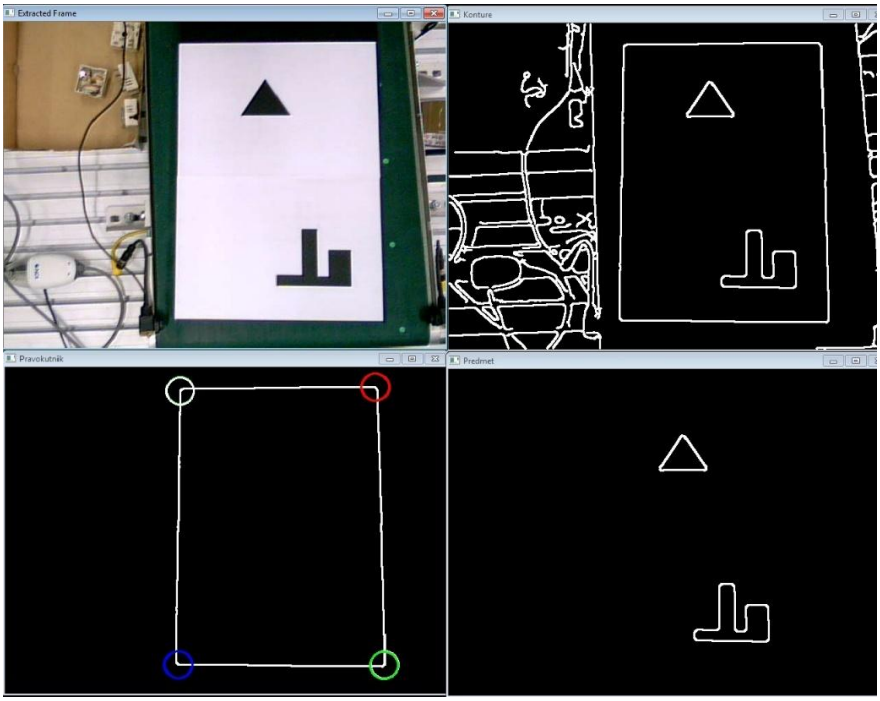


Slika 17. Profil, orijentacija 1



Slika 18. Profil, orijentacija 2

Sustav je u primarnom obliku zamišljen za prepoznavanje pojedinačnih kontura predmeta na radnoj površini. Za prepoznavanje dodatnih kontura unutar radne površine potrebno je izmijeniti način na koji se pristupa hijerarhiji kontura. Potrebno je tražiti konture koje imaju istog roditelja, konturu radne površine. Ako se od više kontura želi istaknuti samo jedna, ostale je moguće eliminirati dodatnim provjerama poput veličine površine ili duljine konture, broja njenih čvorova, pa čak i geometrijskih značajki poput unutarnjih kutova.

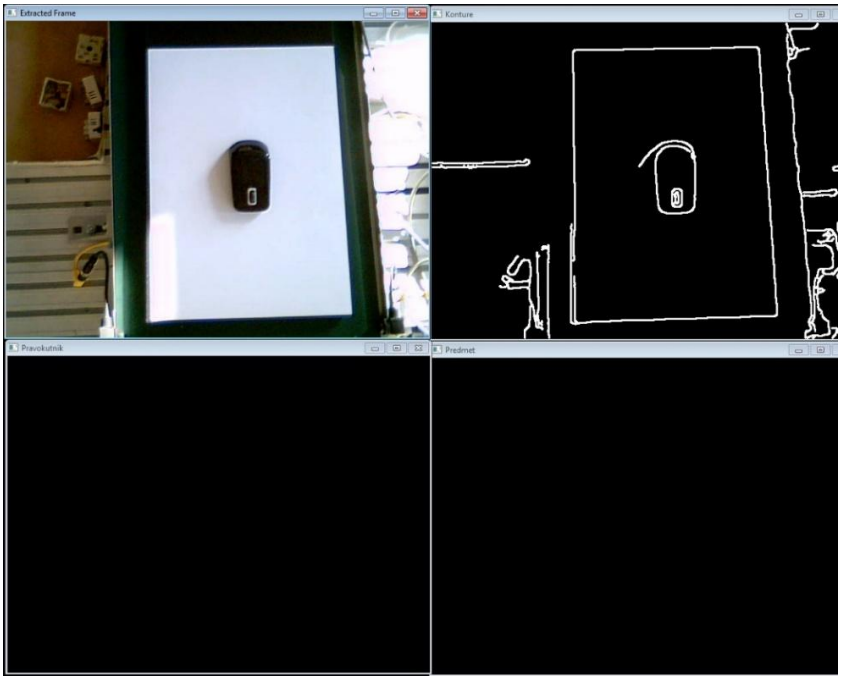


Slika 19. Detektiranje dvije konture istovremeno

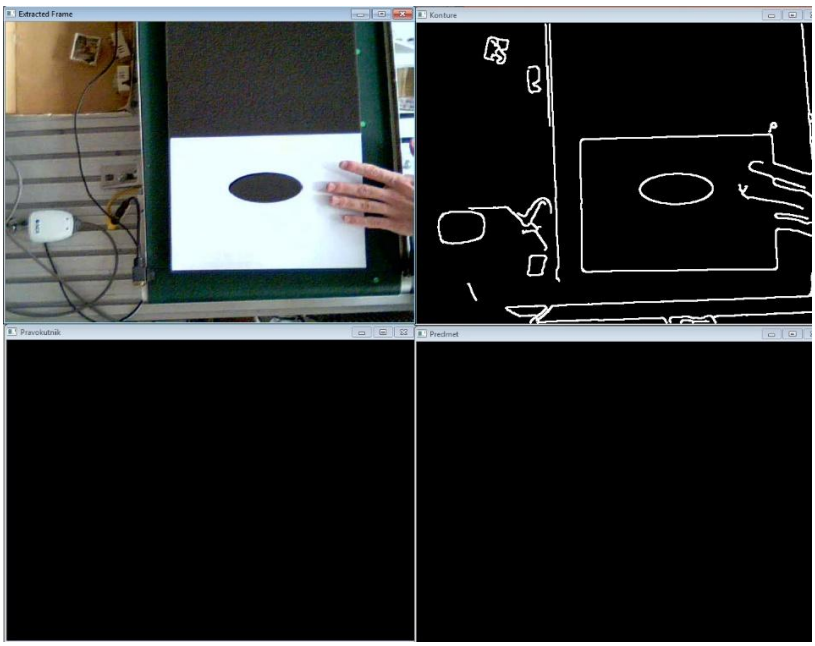
U nastavku su navedeni primjeri kada sustav ne daje očekivane rezultate. Ograničenje sustava u ovakvom obliku je da ne smije biti prepreka na radnoj površini, u suprotnom se neće prepoznati njena kontura. Rezultat toga je da konture unutar radne površine nemaju pravilno definirane hijerarhijske odnose, što onemogućava pristup konturama traženih objekata.

Drugi primjer je neuravnotežena količina osvjetljenja koju je potrebno izbjeći u svakom vizijskom sustavu od kojega očekujemo konzistentne rezultate.

Neadekvatno osvjetljenje i sjena interpretiraju se kao kontrast što rezultira pogrešnom interpretacijom sustava, neuspješnim detektiranjem konture objekta, ali čak i radne površine:



Slika 20. Neadekvatno osvjetljenje



Slika 21. Prepreka

3. Transformacija koordinatnih sustava i komunikacija

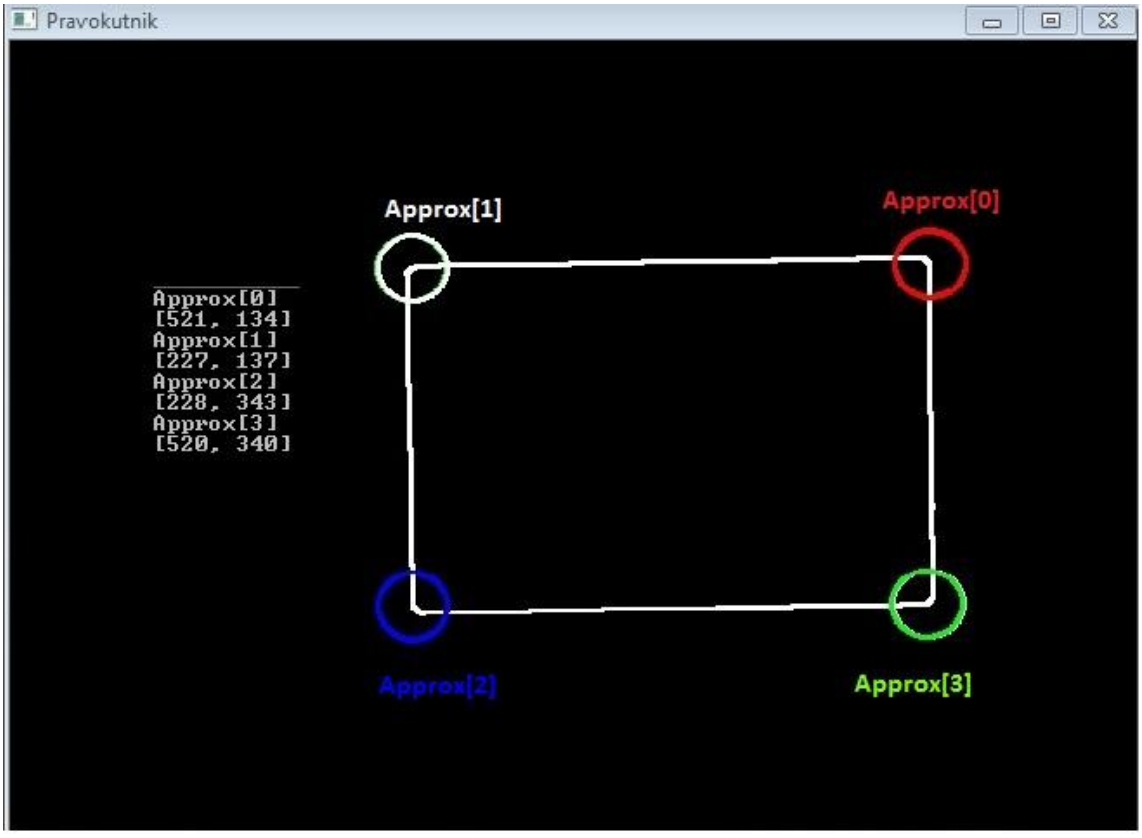
Točke koje opisuju obrise predmeta u koordinatnom sustavu kamere potrebno je komunicirati robotu Fanuc M-3iA kako bi ih mogao pratiti svojom prihvatnicom. Trebalo je uzeti u obzir da su te točke u vizijskom sustavu mjerene u pikselima, dok su koordinate gibanja robota mjerene milimetrima. Međutim, to nije bila jedina prepreka. Koordinatni sustavi nemaju isto ishodište i trebalo je pronaći način da se orijentacija i položaj ta 2 koordinatna sustava usklade.

1. Koordinatni sustav kamere

Iz slike radne površine ne može se zaključiti gdje je ishodište koordinatnog sustava.

Ne samo to, nego postoji razlika visine između mogućih uglova slike i radne površine.

Teško bi bilo pronaći referentnu točku novom koordinatnom sustavu robota i točno pozicionirati vrh prihvatnice u takvo ishodište. Kao logičan izbor nameće se jedan od vrhova radne površine.



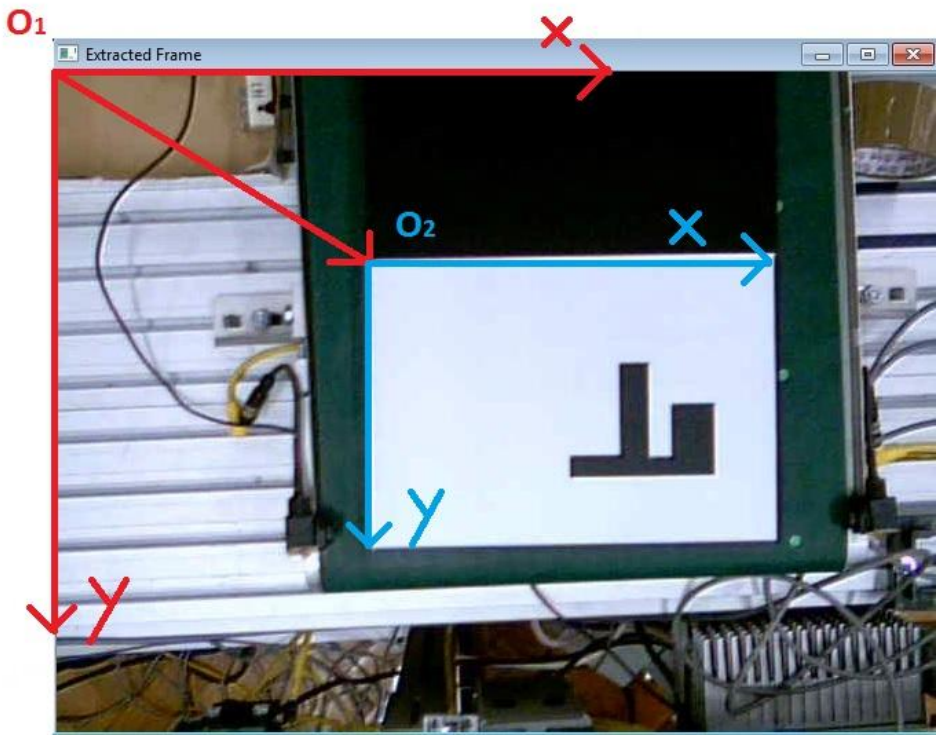
Slika 22. Čvorne točke radne površine

Kako bi se izbjegla nepotrebna promjena predznaka koordinata transformacijom koordinatnog sustava rotacijom[6], vizualizirani su čvorovi raznim bojama i ispisane njihove koordinate. Htjelo se utvrditi najbliži čvor ishodištu koordinatnog sustava kamere, ali i orjentacija koordinatnih osi.

Svaki čvor odabran je kao centar kružnice, s time da se boja kružnice svakog čvora razlikuje.

Za nulti čvor u listi *approx* odabrana je crvena, prvi bijela, drugi plava, te treći zelena boja.

Sada je lako očitati da je 1. čvor u listi najbliži ishodištu koordinatnog sustava kamere, te je on odabran kao novo ishodište koordinatnog sustava.



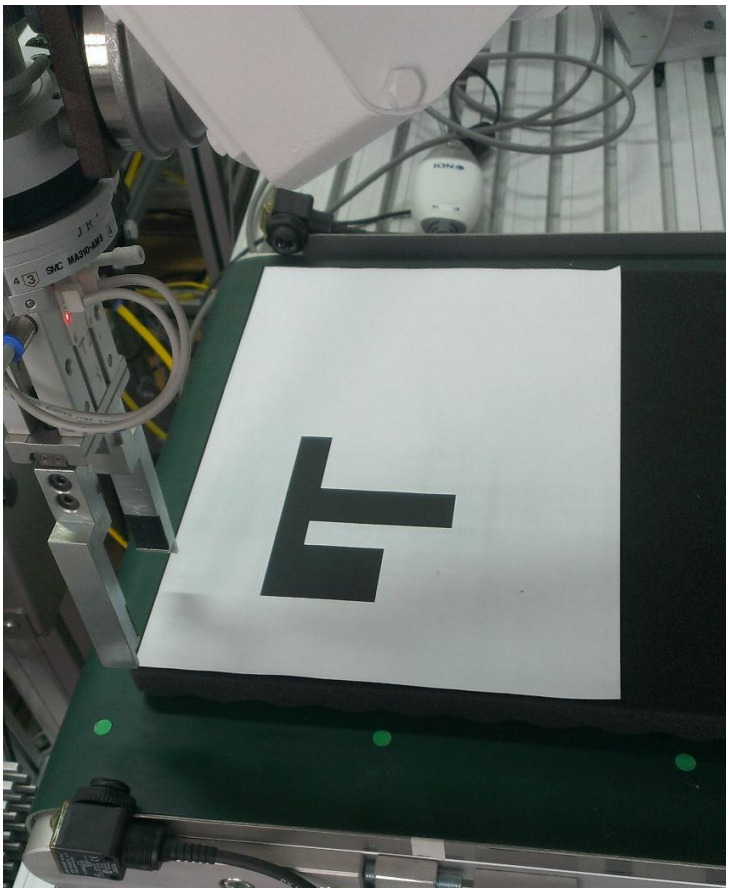
Slika 23. Translacija koordinatnog sustava kamere

Točke obrisa predmeta pohranjene su u vektoru, čiji svaki element ima komponentu x i y. Napravljena je petlja koja od x i y koordinata svake točke konture oduzima iznos novog ishodišta. Zatim taj iznos, koji predstavlja udaljenost pojedine točke od koordinatnog sustava O_2 iz vrijednosti piksela pretvoren je u milimetre. To je sada jednostavno napraviti jer je lako izmjeriti dimenzije površine koju koristimo, a znamo i međusobnu udaljenost čvorova radne površine u pikselima. Kao primjer poslužit će gornja slika koordinatnog sustava. Širina tako orijentiranog papira iznosi 297mm, dok razlika u iznosu piksela čvorova iznosi 196. Ispada da je jedan piksel približno jednak 1.51mm.

2. Koordinatni sustav robota

Slijedi kratak opis postavljanja korisničkog koordinatnog sustava na Fanuc M3-iA robotu detaljnije opisanog u [5].

Unutar korisničkog sučelja privjeska za učenje robota ulazi se u glavni menu, zatim pod stavkom *Setup*, odabire se opcija *Frames*. Primjenila se metoda 3 točke u kojoj se definira ishodište u čvoru radne površine koje se poklapa sa ishodištem transformiranog koordinatnog sustava kamere. Potom se ostvari gibanje u pozitivnom smjeru osi x i y, te robot iz definicje desnokretnog koordinatnog sustava zaključuje u kojem području će biti os z. Visina je konstantna tokom cijelog procesa, tako da ju nije potrebno posebno definirati.



Slika 24. Definiranje koordinatnog sustava robota

Prije upotrebe robota, koordinatni sustav potrebno je aktivirati. Aktivacija se može odraditi na 3 načina, a najpoželjnija je ona unutar programa što nam osigurava da je aktivan koordinatni sustav u kojem su definirane naše točke.

3. Komunikacija sa robotom

Nakon što su usklađeni koordinatni sustavi i točke pretvorene u milimetre, potrebno ih je komunicirati u obliku niza znakova (*string)*. Napravljena je petlja koja u programskom jeziku C++ prolazi kroz svaku točku vektora *contours*, pretvara njen tip podataka iz broja u znak i preko programskog jezika Karel šalje koordinate točaka robotu Fanuc M3-iA. Prvo se šalje koordinata x, zatim y, dok je koordinata z konstantno iznosa 0 jer je riječ o plošnim predmetima i nije potrebno mijenjati visinu prihvatnice. Slijedi isječak koda Karel-a za čitanje točaka i pomicanje prihvatnice.

```
WHILE TRUE DO      -- GLAVNA PETLJA
  BYTES_AHEAD(file_var_C,nb,STATUS)  --čitanje koliko ima bajtova u međuspremniku
  --WRITE('NB:',nb,CR)
  IF nb>3 THEN
   p=0;
    READ file_var_C (xs::0::2, ys::0::2, zs::0::2, ruka::0::2, sp::0::2, hvat::0::2)
    WRITE(  xs::0::2,ys::0::2,zs::0::2,CR)
    CNV_STR_REAL(xs, x9); CNV_STR_REAL(ys, y9);  CNV_STR_REAL(zs, z9);
    CNV_STR_INT (ruka,ruka9);  CNV_STR_INT (sp, sp9);  CNV_STR_INT (hvat, hvat9);

     -- IF hvat9 =1 THEN
       --  RDO[1] = OFF --zatvori hvataljku
        --RDO[2] = ON --zatvori hvataljku
     -- ELSE
       --  RDO[2] = OFF --otvori hvataljku
        -- RDO[1] = ON --zatvori hvataljku
     -- ENDIF
```

```
P02=P01
    P02.x = P01.x + x9 ; P02.y = P01.y + y9 ; P02.z = P01.z + z9;

     IF P02.z < 0 THEN; P02.z = 0;ENDIF
     IF P02.z >300 THEN; P02.z = 300;ENDIF

     IF P02.x < -250 THEN; P02.x = -250;ENDIF
     IF P02.x > 400 THEN; P02.x = 400;ENDIF

     IF P02.y < -700 THEN; P02.y = -700; ENDIF
     IF P02.y > -350 THEN; P02.y = -350;ENDIF
```

Slika 25. Praćenje kontura robotom

4. Zaključak

Vizijski sustav temeljen na Canny algoritmu i hijerarhijama kontura pokazao se kao dobro rješenje pronalaženja kontura objekata nedefiniranih veličina, oblika i orijentacije na radnoj površini. Sliku je svakako prije pronalaženja kontura potrebno obraditi i filtrirati kako bi se povećala točnost i smanjila odstupanja u poziciji kontura. Unatoč primjeni aproksimacijske funkcije prilikom pronalaženja kontura unutar binarne mape izlaza Canny algoritma, uočeno je da nisu sve točke konture jednoznačne, a neke su i redudantne, te je potrebno primijeniti dodatni algoritam aproksimacije za dobivanje optimalnog broja točaka, čime se povećava učinkovitost.

Sustav je primarno modeliran za pronalaženje konture pojedinačnog predmeta na radnoj površini u kontroliranim uvjetima. Ipak, predviđanjem mogućih karakteristika stranih tijela ili neželjenih objekata na radnoj površini može ih se uspješno eliminirati postavljanjem dodatnih uvjeta provjere. U slučaju da postoji mogućnost pojave neželjenih objekata manjih dimenzija na radnoj površini, postavlja se uvjet traženja konture površine veće od definiranog iznosa. Ako znamo geometrijske karakteristike oblika neželjenih objekata, može ih se eliminirati iz mape rubova po broju čvorova.

Nalaze li se dva ili više objekata na radnoj površini, čije konture i koordinate točaka tih kontura želimo pronaći, hijerarhijskom stablu pristupa se na drugačiji način. Ne traži se potomak konture radne površine, već se postavlja uvjet da konture koje tražimo moraju zadovoljiti uvjet vlastitog roditelja. Taj roditelj jedino može biti kontura radne površine jer se konture traženih objekata međusobno nalaze na istoj hijerarhijskoj razini.

Kao i kod svakog vizijskog sustava, osvjetljenje ima vrlo bitnu ulogu u konzistentnosti rezultata, pa se svakako preporuča dobro osvjetljena podloga u svrhu minimiziranja šuma.

Rezultati dobiveni vizijskim sustavom opisanim u ovom radu su zadovoljavajući, iako bi se mogli poboljšati. Korištenjem kvalitetnije kamere imali bismo veću rezoluciju slike što bi značilo veću preciznost prilikom konverzije koordinati točaka u milimetre, dok bi kvalitetnija leća minimizirala smetnje i omogućila oštriji prikaz slike. Kalibracija kamere pružila bi vjerniji prikaz stvarne slike, čime bi se dodatno smanjila odstupanja pronađenih kontura. U realnom procesu automatiziranog brušenja koriste se senzori sile, čija implementacija regulira udaljenost radnog alata od ruba predmeta i osigurava međusobni kontakt. Trebalo bi eksperimentalno utvrditi kolika odstupanja vizijskog sustava su unutar tolerancija takvog procesa.

Literatura:

[1] Allain A.: Jumping into C++ by Alex Allain & Sample Code, 2014.

[2] http://docs.opencv.org

[3] http://sourceforge.net/projects/opencvlibrary/

[4] Laganière R.: OpenCV 2 Computer Vision Application Programming Cookbook, 2011.

[5] Švaco M., Šekoranja B., Jerbić B.: Programiranje FANUC robota – osnove, Zagreb, 2011.

[6] Šurina T., Crneković M.:Industrijski roboti, , Zagreb, 1990.

[7] Canny edge detector - Wikipedia http://en.wikipedia.org/wiki/Canny_edge_detector, 14.12.2014

[8] D L Baggio: Mastering OpenCV with Practical Computer Vision Projects, 2012.

Prilog

Izvorni kod programa u C++ jeziku.

```
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv/cv.h>


static double angle(cv::Point pt1, cv::Point pt2, cv::Point pt0)
{
double dx1 = pt1.x - pt0.x;
double dy1 = pt1.y - pt0.y;
double dx2 = pt2.x - pt0.x;
double dy2 = pt2.y - pt0.y;
return (dx1*dx2 + dy1*dy2)/sqrt((dx1*dx1 + dy1*dy1)*(dx2*dx2 + dy2*dy2) + 1e-10);
}




int main(int argc, const char** argv)
{

	cv::VideoCapture capture;
	capture.open(0);
	if (!capture.isOpened())
		return 1;
	bool stop(false);
	bool singlestep = false;
	static bool once1 = true;
	static bool once2 = true;

	// Varijable

	int thresh = 50;

	int c;
	bool rawImage;
	cv::Mat Frame,Image_Gray;
	cv::Mat Konture;
	cv::Mat canny_output;
	cv::vector<cv::vector<cv::Point> > contours;
	cv::vector<cv::Vec4i> hierarchy;
	cv::namedWindow("Extracted Frame",CV_WINDOW_AUTOSIZE);
	cv::namedWindow("Pravokutnik",CV_WINDOW_AUTOSIZE);
	cv::namedWindow("Predmet",CV_WINDOW_AUTOSIZE);
	std::vector<cv::Point> approx;
	cv::Mat erode_output;
	cv::Mat pravokutnik,pyr,Image,Image_Blur;
```

```
	cv::Scalar color = cv::Scalar( 255, 255, 255 ); // boja kontura (bijela)

	// Najveća površina pravokutnih kontura
	int largest_area = 0;
	int largest_contour_index=0;



	cv::namedWindow("Konture",CV_WINDOW_AUTOSIZE);

	// int delay = (1000/rate);


	for(;;)
	{
		if ( !stop )
		{
			rawImage = capture.read(Frame);
			if (!rawImage)
				break;
		}
		cv::imshow( "Extracted Frame",Frame);

		if ( singlestep )
			stop = true;

		// Trazenje kontura ako su singlestep i rawImage = true
		if( rawImage && singlestep)

		{
		// Filtriranje slike prije canny algoritma
		cv::pyrDown(Frame, pyr, cv::Size(Frame.cols/2, Frame.rows/2));
		cv::pyrUp(pyr, Image , Frame.size());

		cv::cvtColor(Image, Image_Gray, CV_BGR2GRAY);
		cv::medianBlur(Image_Gray, Image_Blur, 7);

		Canny( Image_Blur, canny_output, 0, thresh*4, 3 );

		//cv::erode(canny_output, canny_output, cv::Mat(), cv::Point(-1, -1), 2, 1,
		1);
	 //cv::dilate(canny_output, canny_output, cv::Mat(), cv::Point(-1, -1), 2, 1, 1);

		findContours( canny_output, contours, hierarchy, CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE, cv::Point(0, 0) );
		cv::Mat drawing = cv::Mat::zeros( canny_output.size(), CV_8UC3 );
		cv::Mat pravokutnik = cv::Mat::zeros( canny_output.size(), CV_8UC3 );
		cv::Mat predmet = cv::Mat::zeros( canny_output.size(), CV_8UC3 );
		for( int i = 0; i< contours.size(); i++ )
		{



			drawContours( drawing, contours, i , color, 2, 8, hierarchy, 0,
cv::Point() );
```

```
                    // Provjera oblika svake konture
                    // Approximate contour with accuracy proportional
                    // to the contour perimeter
                    cv::approxPolyDP(cv::Mat(contours[i]), approx,
cv::arcLength(cv::Mat(contours[i]), true)*0.02, true);


                    // Ako kontura ima 4 čvora(pravokutnik)
                    if (approx.size() == 4 && (std::fabs(cv::contourArea(contours[i])) >
100000))
                    {


                           // Number of vertices of polygonal curve
                           int vtc = approx.size();
                           // Get the cosines of all corners
                           std::vector<double> cos;
                           for (int j = 2; j < vtc+1; j++)
                           cos.push_back(angle(approx[j%vtc], approx[j-2], approx[j-1]));
                           // Sort ascending the cosine values
                           std::sort(cos.begin(), cos.end());
                           // Get the lowest and the highest cosine
                           double mincos = cos.front();
                           double maxcos = cos.back();
                           // Use the degrees obtained above and the number of vertices
                           // to determine the shape of the contour
                           if (vtc == 4 && mincos >= -0.15 && maxcos <= 0.3)
                           {
                                  /* Traženje najveće konture (podloge/okvira)
                                  double a = cv::contourArea(contours[i],false);
                                  if (a > largest_area)
                                  {
                                  largest_area = a;
                                  largest_contour_index = i;
                                  }*/

                                  drawContours( pravokutnik, contours, i , color, 2, 8,
hierarchy, 0, cv::Point() );
                                  // Ispis točaka konture podloge

                                  /*if ( once1 )
                                         {
                                                std::cout<<contours[i]<<std::endl; //
točke konture okvira
                                                std::cout<<hierarchy[i]<<std::endl; //
hijerarhija konture okvira
                                                std::cout<<hierarchy[i][2]<<std::endl;
                                                once1 = false;
                                         }
                                  */
                                  int chedo = hierarchy[i][2]; // čedo konture podloge



                                  if ( hierarchy[chedo][2] < 0) // ako čedo nema čedo,
ispiši čedo
                                  {
```

```
                                        drawContours( predmet, contours, chedo , color,
2, 8, hierarchy, 0, cv::Point() );
                                        if ( once1 )
                                        {
                                               std::cout<<contours[chedo]<<std::endl; //
točke konture
                                               once1 = false;
                                        }

                                        // contours[chedo].size();

                                 }
                                 else // ako čedo ima čedo, iscrtaj čedino čedo
                                 {
                                        drawContours( predmet, contours,
hierarchy[chedo][2] , color, 2, 8, hierarchy, 0, cv::Point() );
                                        if ( once1 )
                                        {

       std::cout<<contours[hierarchy[chedo][2]]<<std::endl; // točke konture
                                               once1 = false;
                                        }
                                 }

                           }

                    }

             cv::imshow("Konture",drawing);

             cv::imshow("Pravokutnik",pravokutnik);

             cv::imshow("Predmet",predmet);

             }

             }

             c = cvWaitKey(10);
       c = tolower(c);

             // End processing on ESC, q or Q
       if(c == 27 || c == 'q')
           break;
       switch( c )
       {
       case 'p':
           stop = !stop;
           break;
       case 's':
           singlestep = true;
           stop = false;
                    once1 = true;
                    once2 = true;

           break;
       case 'r':
           stop = false;
```

```
            singlestep = false;
            break;
              }


        }
        capture.release();
        cv::destroyAllWindows()


}
```